CORE
Metadata, citation and similar papers at core.ac.uk
Provided by ethesis@nitr

Finding Clones in UML Sequence Diagrams and Web Services

Anil Kumar Mahalik

Roll. 213CS3187

under the supervision of

Prof. Durga Prasad Mohapatra

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela – 769008, India

Finding Cloness in UML Sequence Diagrams and Web Services

Dissertation submitted in

June 1

to the department of

Computer Science and Engineering

of

National Institute of Technology Rourkela

in partial fulfillment of the requirements

for the degree of

Master of Technology

by

Anil Kumar Mahalik

(Roll. 213CS3187)

under the supervision of

Prof. Durga Prasad Mohapatra

Department of Computer Science and Engineering

National Institute of Technology Rourkela

Rourkela – 769 008, India

Computer Science and Engineering

National Institute of Technology Rourkela

Rourkela-769 008, India. www.nitrkl.ac.in

June 1, 2015

Certificate

This is to certify that the work in the thesis entitled *Finding Clones in UML sequence Diagrams and Web services* by *Anil Kumar Mahalik*, having roll number 213CS3187, is a record of an original research work carried out by him under my supervision and guidance in partial fulfillment of the requirements for the award of the degree of *Master of Technology* in *Computer Science and Engineering Department*. Neither this thesis nor any part of it has been submitted for any degree or academic award elsewhere.

Dr. Durga Prasad Mohapatra

Associate Professor

Department of CSE

NIT, Rourkela

Acknowledgment

First of all, I must convey my deep sense of respect and gratitude to my mentor, supervisor and a real human being Prof. Durga Prasad Mohaptra , for his uninterrupted motivation and support in my project work and thesis organization. It was my fortune to work under his guidance. I will remain ever grateful near him in improving me in all respect. I am touched with his soft words and kind heartedness and those are my treasure with which I will move ahead in my life journey.

I would like to express special thanks to my HOD Prof. Shantanu Kumar Rath for being a constant source of inspiration since my first day at the Department. I must not hesitate to admit that he has many things in his personality to be learnt. I have been very much impressed and influenced by his personality. His coerce and consultation has rectified me a lot.

I must thank all other faculty members,the Ph.D. research scholars, academically seniors and juniors for their odd time help in all respect. Their appreciations and criticize have made me the current me.

Though old, but always fresh my lab mates are required to be acknowledged more for beating with my heart bits and making the lab stay more productive and pleasant. I am facing difficulty in taking the names of all my friends from CSE department, outside the department and even out of NIT Rourkela tertiary on this single page of acknowledgement. But from the core of my heart i will stay indebted near them for everything they have provided so far.

I must thank my parents for the toil they have shouldered in bringing up and educating me, to my family members and relatives for tolerating me so far and for making me eligible and sociable with their help and advice.

I must thank the almighty for gifting me many wonderful persons in my life and a wonderful society to live. Beyond all these unlimited matters I thank the most to my fate and fortune for making all the right things happen at right point of time.

Anil Kumar Mahalik

Abstract

UML Sequence diagram is used to express the behavioral aspect of Design Models. Finding similarity in UML Sequence diagram is required in order to avoid redundancy of similar fragments, thereby reducing the resource consumption. The units chosen for comparison in UML Sequence diagrams are fragments representing conversation. A conversation is nothing but a Behavioral Execution Specification (BES) having sequences of Message Occurrence Specifications (MOS) and hence caused events embedded within it. For comparing various conversations, frequently occurring patterns are detected and reported as clones. Similarity reporting in our work has been accomplished by a text based clone detector Nicad which basically performs on XMI codes. Nicad can not find similarity directly from XMI codes because the XMI representation of sequence diagram is very complex to understand and process. It needs a step by step procedure like Identification, Consolidation and Contenxtualization to add clear meaning to the context. The restructuring is done with the help of Txl a transformation language. It has its own grammar and transformation rules to redefine the XMI codes to a pretty printing form. Nicad can work only on the Contextualized XMI codes.

We have utilized same tool Nicad in finding similarity in various WSDL files as WSDL itself is written in XMI. As we are approaching towards an era of Service Oriented Architecture (SOA), services are the most important things to be concentrated. In web services, the unit of comparison is the service operations enclosed inside the WSDL file. We have used the same Txl grammar and Transformation rules to convert the complex WSDL to well structured contextualized operation descriptions and enclose them inside a self contained unit. In order to enhance the ease of service identification and Service discovery, similarity in Web Services need special attention. During the down time or heavy traffic period, if a service is found unavailable, we can get our work done lending the service from another service provider identified previously. This is called web service tagging.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Complex dynamic interactions of interactive systems such as web applications are presented using UML sequence diagrams. Lifelines are used to present the user, the browser, the server, the back end database and various tiny processes(threads) within them concurrently. Sequence diagrams in forward engineering used to specify intended behavior [5, 6] and in reverse engineering to observe and document the actual behavior.

Starting with nothing more than a set of inter-connected static pages, after for nearly two decades world wide web has evolved into something much more. Whether it is social network like face book and twitter or e-commerce site like flipkart or snapdeal web has become a platform for rich Internet applications. Creating new applications and composing new and more complex services are the current days demands. Hence, there is a pressing need of APIs in web applications to call their services. Tagging and categorization are required to find services in order to overcome the difficulty of finding the right services keeping in eyes the exponential growth of Service now-a-days [10, 12, 13].

1.1 Motivation

For every complex interactive web applications [14], reverse engineered sequence diagrams are often very large to analyze by hand. It is too difficult to identify repeated sequences of behavior between components.

WSDL is used to describe huge no of web service.As they are complex in structure, it is difficult to find similarity in them. Description of all the operations are contained in the WSDL Document. Separated pieces of the operations are scattered through the file. So it is very challenging to find the similarity after comparing the operations.

So, in this thesis, we propose an automated technique to identify similar interactions in the sequence diagrams reverse model using near miss code clone detector, Nicad. The same Nicad can be utilized to find similarity in operations in various WSDL documents.

We follow an text based method, working the level of XMI, the standard exachange language for UML. WSDL as usual always represented in XMI. Hence Nicad performs his processing on XMI and provide the extracted clones.

1.2 Contributions

This thesis makes the following contributions-

1. First, we describe a new method of identifying near miss clones in sequence diagrams. This approach is scalable to identify clones in sequence diagram fragments in a very complex reverse engineered Sequence diagrams.

2. We have constructed and identified those conversations as a number of self possessed conversational units. Units of comparison for clone detection are those conversational units.

3. We have proposed an extension to the architecture of Nicad clone detector is proposed to include a phase at the beginning that permits Txl Transformer to be applied on the actual source.

4. We have applied a technique to modify the original source code fragment to add contextualized informations keeping in eye the need of proposed modification.

5. We have applied the same method of finding similarity in UML sequence diagram in finding similarity in web services also.

1.3 Organization of the Thesis :

In Chapter 2, we introduce some of the basic concepts to cope with UML sequence diagrams and web service description languages(WSDL). Various parts and portions of sequence diagram are described in details. The structure of WSDL Document is clearly depicted in this chapter. A Basic understanding of Nicad Clone detector architecture and Txl transformation language Grammar and Transformation rules are described at a glance.

In Chapter 3, related works previously done on detecting clones have been discussed thoroughly.

In Chapter 4, the step by step procedure to find the similarity in sequence diagram fragments is stated. Each and every phase starting from the Identification to clone extraction phase is told briefly.

In Chapter 5, the process of finding similar operations across different web services using Nicad is mentioned. The Txl grammar and transformation rule to get the well structured view of WSDL document is described. A comparison between contextualize and non-contextualized clones and clone classes is presented in the form of a table.

In Chapter 6, the summary of our work of the whole thesis and scope for future improvement is written .

Chapter 2

Background Details

2.1 Basic Definitions

In this chapter we will discuss some basic definitions of Clone detection which will help to understand clone detection in a broader sense. Again we will discuss the process of Clone detection in Sequence diagrams and web services .

2.1.1 UML Sequence Diagrams

UML sequence diagrams (SDs) are 2-Dimensional graphical models used to represent the interaction between various objects or actors in a system, encoding the order in which events and message interactions between the actors occur. They are mainly used to model the behavior of web applications and other interactive applications where the sequencing of interactions over time needs to be specified.

Figure 2.1 shows an example highlighting the main elements in a basic sequence diagram. These include Lifelines, Messages, Behavior Execution Specification (BES), Message Occurrence specification (MOS), Events, Classes. Other elements such as events, Classes, Properties etc. are not shown in the figure for readability.

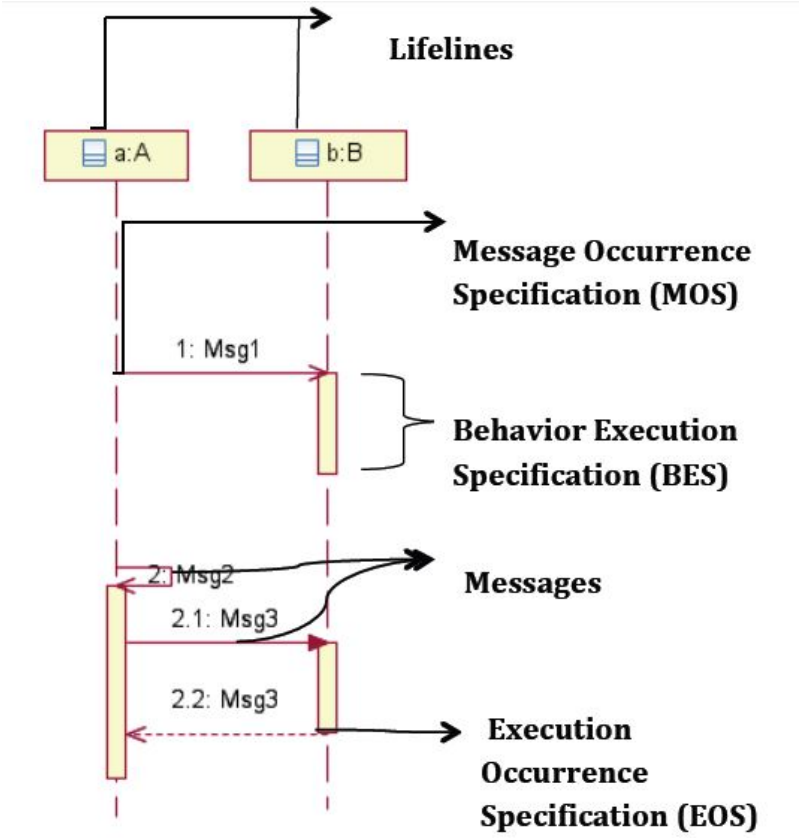

Figure 2.1: Elements of a basic sequence diagram

2.1.2 Web Services

The World Wide Web Consortium (W3C) defines a web service as a software system designed to support interoperable machine-to-machine interaction. Put simply, a web service is an application running on a server that has an interface allowing it to be called by a client.

Web Service Description Languages(WSDL)

In order to be used by client applications, web services must provide some sort of interface description to the application developers. The interface description language depends on the architecture of the web service being described in Figure

2.2.

The easiest method of describing web services, no matter the architecture, is using prose, because an English description of the interface will likely already exist from the design process. The problem with this method is that there is no standard; anything goes. This can result in very poor documentation, making the service difficult to use. It also means the service is difficult to discover, and code stubs cannot be generated automatically to invoke the services operations.

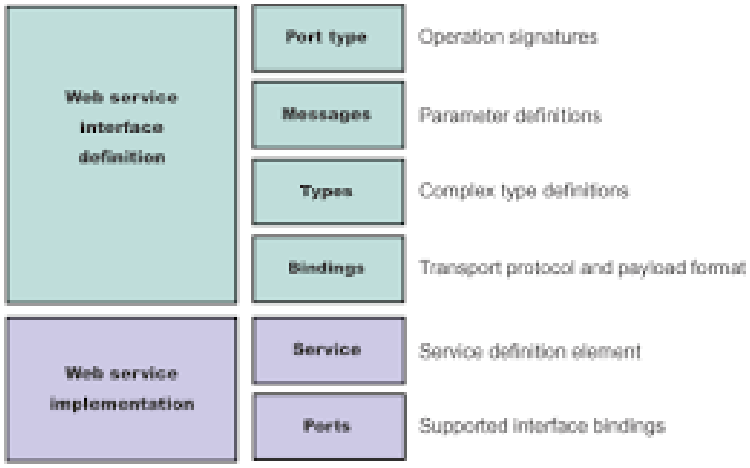

Figure 2.2: Structure of a WSDL description.

2.2 Basic concepts

2.2.1 SD XMI representation

Sequence diagrams can be represented in text using XMI representation, the XML-based standard interchange format used for exchanging models between various modelling tools. Interactions in a sequence diagram are represented in XMI format as "fragments", as shown in Figure 2.1. Every element of a sequence diagram has an XMI Id identified by "xmi:id" and a set of attributes that shows

the relationship of this element with the others. The attribute values have been renamed to aid comprehension.

Sequence diagram model-clone detection entails discovering similar or identical sequences of behavioural interaction ("conversations"). Unlike source code, which is represented as linear text, models are typically represented visually, as box-and-arrow diagrams. Model clones can thus be thought of as similar sub graphs of these diagrams.

2.2.2 Anatomy of a WSDL Description

Operation descriptions that a specific web services offer are kept inside a WSDL file. Basing on the perspective of the operation those descriptions are dissolved in slices and clustered together. These slices, when related with each other construct the operation description. These slices can be categorized into five different sub-categories as follows -types, messages, port types, bindings and services.

Types- Basically, XML Schema is the used medium for interchanging of data between a querying client and a providing web service. This is carried out by type definition defined in the type section. Type section may have single or multiple no of the schema defined inside or referred (.xsd file) explicitly outside the type section. Object possessing other elements are termed as ComplexTypes and are defined inside the schema. Type Room having 2 integers indicating roomID and no of beds and a Boolean (Y/N) to refer smoking or non-smoking in our hotel reservation example as shown. Part referenced elements in the message segments are also defined by the Schema.

Messages- The information corresponding to the input, output and faults of each operation is defined in the message segment. Parts contain the separated input, output and faults referring to the elements in the type segment. Parts

have constraints of operations referencing most of the time to the lements in type segment possessing the constraints. Part name body is inside the message ReserveRoomRequest referring ReserveRoomRequest element of the type segment. The constraints contained in the element operation are payment and room.

Port Types- Single or multiple operations composing the web service are put inside the port type section of a WSDL document. Depending on the variety of message occurs input, output or fault message are carefully kept inside these operations. For example, both input and output element to be put in case of a request-response message. These elements are referred to all other places in the file where they are defined. Operation ReserveRoom contains single input, output and fault those refer to previously defined message section above in our Hotel reservation example.

Bindings- A message pattern and procedure for a port type is described within Bindings section. Simple Object Access Protocol (SOAP) is the frequently used procedure for communication.

Services.- A group of ports are mentioned in the service segment. Ports are the endpoints and place of communication indicating an address for a binding.

2.3 Source Transformation

Source transformation has been used in many software engineering tasks. It involves transforming the original source text to another desirable form for the task at hand. Some examples of its application include translating from one source code language to another, design recovery, and transforming legacy code to a new structured code design. A number of source transformation tools exist including TXL. Various examples of TXLs application in source transformation tasks are provided on the TXL web page.

In our case, due to the flat structure of the XMI sequence diagram representation, it was necessary to transform or restructure it, and gather all elements of a conversation into a conversational unit to bring context. This transformation of the input XMI text to a desired form was done using TXL. We briefly describe TXL and some examples of its capabilities in the remainder of this section.

2.3.1 The TXL Source Transformation Language

TXL is a hybrid functional/rule-based programming language designed to support software analysis and source transformation tasks. The TXL processor works in three distinct phases:

1. **Parse Phase**: In this phase the input is tokenized, and a parse tree generated based on the TXL grammar defined for the input.

2. **Transform Phase**: In this phase, TXL takes the parse tree and produces the desired output based on the rules and functions.

3. **Unparse Phase :** During this phase, it unparses the transformed output and generates the output text based on the formatting specified in the grammar.

2.3.2 TXL grammar

A TXL grammar is a specification for the language of the input source text, specified as a directly interpreted Backaus Naur Form grammar, in context-free ambiguous form. Nonterminal and terminals make up the TXL grammar. TXL grammar begins with a program definition - a special nonterminal that defines structure of the whole input source. Nonterminals define the units with which the source input is constructed. TXL defines a number of built-in nonterminals such as [id], [stringlit],[charlit], and [number], which represent identifiers, strings, characters, and unsigned integer and real numbers respectively.

A nonterminal X is specified in a define X.. end define block. In our case, we have defined our input program to be a sequence of zero or more of [xmi_element] type as shown below

define xmi_element [general_xmi_element] end define define general_xmi_element < [opt xmi_colon] [id] [repeat tag_attribute] /> — < [opt xmi_colon] [id] [repeat tag_attribute]> [xmi_element*] < / [opt xmi_colon] [id] > end define

2.3.3 TXL Transformation Rules

The second component of a TXL program, uses rules and functions to perform the desired source transformation. A TXL rule specifies the type of nonterminal in the parse tree, for which it searches for the pattern specified to be replaced with the desired replacement. TXL parses the input as described by the grammar definition and applies the rules recursively to the input until it fails and produces the transformed output. That is, every match of the type in the parse tree is matched for the pattern and transformed with replacement. For a function, only the first occurrence of the pattern in replaced. The output generated is formatted as defined in the grammar.

Rules are specified within the block rule ruleName .... end rule. A function is defined in a similar way with the keyword rule replaced by keyword function.

The type of the pattern and the replacement must be the same. Rules also use the deconstruct X to break up the patterns into sub-patterns or the deep construct deconstruct X* to search inside the nonterminal X, for a specific pattern.

2.4 NiCad:The clone detector

NiCad is a flexible TXL-based hybrid language-sensitive / text comparison software clone detection system developed by James R. Cordy and Chanchal K. Roy.

NiCad was designed for the automated detection of near-miss intentional clones. It is an easy-to-use powerful command line tool. NiCad 3.5 is used in our experiments. NiCad has successfully been used in many projects for finding clones in languages such as C, Java, WSDL documents, as well as in graphical Simulink models. NiCad takes, as input, the source (in text form or models represented in textual form) to be analyzed along with configuration files which specify the filtering, threshold and type of normalization to be applied on the input source.

NiCad extracts all potential clones specified by the granularity which are normalized (consistent renaming, blind renaming, filtering), if necessary, to eliminate any unwanted differences to make the comparison process more precise and accurate. Normalization improves the clone results by identifying near-miss clones which would otherwise be not reported. NiCad then compares the extracted potential clones line-by-line using an efficient implementation of the Longest Common Subsequence algorithm.

There are two sets of results generated by NiCad, each reported in both HTML and XML formats. First, the results of the comparison are reported as clone pairs that differ in number of lines upto the specified difference threshold. Second, the clones in the input source are grouped into clone classes. Each clone class contains all the clones in the input which are similar and differ in number of lines only upto the specified difference threshold. Both formats contain source of the clones, specifying the degree of similarity, start and end line numbers of the clones found and the size of the clones.

NiCad provides the ability to find clones at various granularities (classes, functions,blocks, statements, etc.), with varying degrees of near-miss similarity (e.g., 70, 80, 90 or 100% similar). It can be used either to find clones within a system, or cross-clones between two different systems.

NiCad is a very scalable, and powerful clone detector and the results of the clone detection have high recall and precision.

2.5 An example illustrating Txl Grammar and Txl Transformation Rules

We state here the Txl grammar and transformation rules to extract the M.Tech. students from no of students.

2.5.1 An Example of Txl Grammar for Students

tokens
emailaddress " $[-\ i.] + @ [-\ i.] +$ "
end tokens
define program
$[student\_list]$
end define
define student_list
$[SPOFF]' <' \ studentList > [SPON]\,[NL]\,[IN]$
$[repeatstudent]\,[EX]$
$[SPOFF]' < /' studentList > [SPON]\,[NL]$
end define
define student
$[SPOFF]' <' \ student > [SPON]\,[NL]\,[IN]$

$[repeatstudent\_attr^{+}]\,[EX]$
$[SPOFF]' < /' student > [SPON]\,[NL]$
end define
define student_attr
$[name]$
$|\,[degree\_type]$

| [*email*]
end define
define name
[$SPOFF$]′ <′ *name* > [$SPON$] [*repeatid*]
[$SPOFF$]′ < /′*name* > [$SPON$] [NL]
end define
define degree_type
[$SPOFF$]′ <′ *degreeType* > [$SPON$]
[*id*]
[$SPOFF$]′ <′ *degreeType* > [$SPON$] [NL]
end define
define email
[$SPOFF$]′ <′ *email* > [$SPON$] [*emailaddress*]
[$SPOFF$]′ < /′*email* > [$SPON$] [NL]
end define

2.5.2 Txl transformation rules for creating a list of MTech Students

include "StudentList.grm"
redefine program ...
| [$SPOFF$]′ <′ *MTechStudents* > [$SPON$] [NL] [IN]
[*repeatstudent*] [EX]
[$SPOFF$]′ < /′*MTechStudents* > [$SPON$] [NL]
end redefine
function main
replace [*program*]
P [*program*]
deconstruct P
′ <′ *studentList* >
Students [*repeatstudent*]

$' < /' studentList >$

$constructMTechs\,[repeatstudent]$

$\_\,[getMTechseachStudents]$

$constructNewP\,[program]$

$' <' MTechStudents >$

MTechs

$' < /' MTechStudents >$

by

NewP

end function

$functiongetMTechsStudent\,[student]$

$replace\,[repeatstudent]$

$MTechStudents\,[repeatstudent]$

$constructDegreeType\,[repeatdegree\_type]$

$\_\,[Student]$

deconstruct DegreeType

$' <' degreeType > MTech' < /' degreeType >$

$\_\,[repeatdegree\_type]$

by

$MTechStudents\,[.Student]$

end function

2.5.3 Input Program to the parser

<studentList>

<student>

<name>Anil Mahalik</name>

<degreeType>MTech</degreeType>

<email>amahalik89@gmail.com</email>

</student>

<student>

<name>Deepak Nayak</name>

<degreeType>PhD</degreeType>

<email>depakranjan@gmail.com</email>

</student>

<student>

<name>Arun sahani</name>

<degreeType>MTech</degreeType>

<email>sahani.arun@gmail.com</email>

</student>

</studentList>

2.5.4 Output generated after parsing

<MTechstudent>

<student>

<name>Anil</name>

<degreeType>MTech</degreeType>

<email>amahalik89@gmail.com</email>

</student>

<student>

<name¿Arun</name>

<degreeType>MTech</degreeType>

<email>sahani.arun@gmail.com</email>

</student>

</MTechStudents>

2.6 Summary

The NiCad clone detector has been successfully used in finding clones in many source code languages. It is a text based clone detector which requires a specified granularity which occurs naturally in most source code languages. Examples of granularity in source code languages include functions, blocks , statements, or even a class level granularity for Object Oriented languages. Through various steps we have been able to find extracted clones reported by NiCad Clone detector .

Chapter 3

Related Works on Clone Detection

Roy et al. [1] have used NiCad tool that needs a original directory or directories in which presence of similarity is assumed and a configuration file indicating the normalization and filtering to be performed as input and provides output results in both XML form for ease of analysis and HTML form for ease of browsing.

Liu et al. [2] have suggested to convert the two dimensional Sequence diagram to one dimensional array. From the array a suffix tree could be formed. Finally with the common prefixes they have detected the clones.

Martin et al. [3] have introduced the idea of contextual clones clones that can only be found by augmenting code fragments with related information referenced by the fragment to give it context and then finding clones using NiCad.

Dong et al. [4] describe the algorithms underlying the Woogle search engine for web services. Woogle supports similarity search for web services, such as finding similar web service operations and finding operations that compose with a given one.

J. Cordy [11] has provided a clear understanding about Txl parser. He has clearly defined how to develop Txl grammar and Txl transformation rules as well for any

web based applications.

Komondoor et.al [8] have used program slicing techique to extract code clones as clones are always considered harmful because of their more resource consuming nature.

Gauthier et al [9]have used clone detection to find clusters of security sensitive code in open source PHP web applications. With the assumption that syntactically similar clones should have similar access control privileges. They hypothesize that clones that do not follow this assumption violate security privileges and report them as security discordant clones.

Strrle et. al [7] has told about challenges and possibilities in clone detection in all types of UML domain models. His work is based on an earlier work on model matching and model querying . He observes that UML models are loosely connected graphs of heavy nodes. He implements graph matching and represents model elements as facts and models as a set of facts in Prolog, then encode Prolog rules to find clones using various similarity heuristics of model elements. Clone detection algorithm and the evaluation of the heuristics, is implemented as MQlone tool, a plugin in MagicDraw UML CASE tool which reports the clones to the user.

Chapter 4

Clone Detection in UML Sequence Diagrams

We define clones in SDs to mean repeated patterns of similar or identical interaction elements in complete conversations. A conversation is defined as a sequence of message interactions between various actors during a specific period of time (i.e, within a BES). In our thesis we are interested in identifying repeated conversations, and we define clones in SD from this perspective. Code clones and model clones have been classified according to the types they can identify. Using a similar format we define clones in SD according to the types they can identify as follows.

1. **Type 1 (exact):**clones have identical interaction elements in a conversation except for variations in visual presentation, layout and formatting.

2. **Type 2 (renamed):** clones are Type 1 clones that are only renamed (i.e., Type 1 clones that additionally differ only in identifiers and constant values).

3. **Type 3 (near miss):** clones that have small differences such as additions, deletions, or modifications in the interaction elements that are part of a conversations. Type 3-1 clones are identical Type 3 clones without renaming. Type 3-2 clones are Type 3 clones with differences and renaming.

4.1 Approach to clone detection

Our approach to clone detection consists of four main stages as shown in the Figure 4.1.

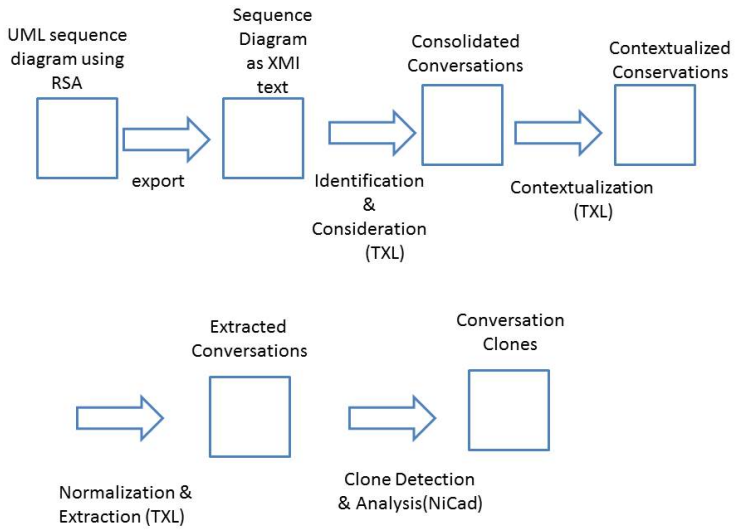

Figure 4.1: Steps involved in our approach

4.1.1 Get the XMI representation of the sequence diagram

In the flat structure of the XMI sequence diagram representation, there is little locality; fragments and elements of sub-conversations are spread across the text. XML attributes and the order of elements are used to reference and implicitly group related elements. Behaviour Execution Specifications (BESs), for example, (in figure a BES element highlighted in green), reference the lifeline they are part of using the covered attribute, and the sequence of messages and events of the conversation using the start and finish attributes, and these elements in turn refer to their parts in similar fashion. In order to restructure this scattered representation for comparison purposes, we need to recursively gather the referenced and related elements of the BES conversations together and organize them explicitly into the structures they

represent.

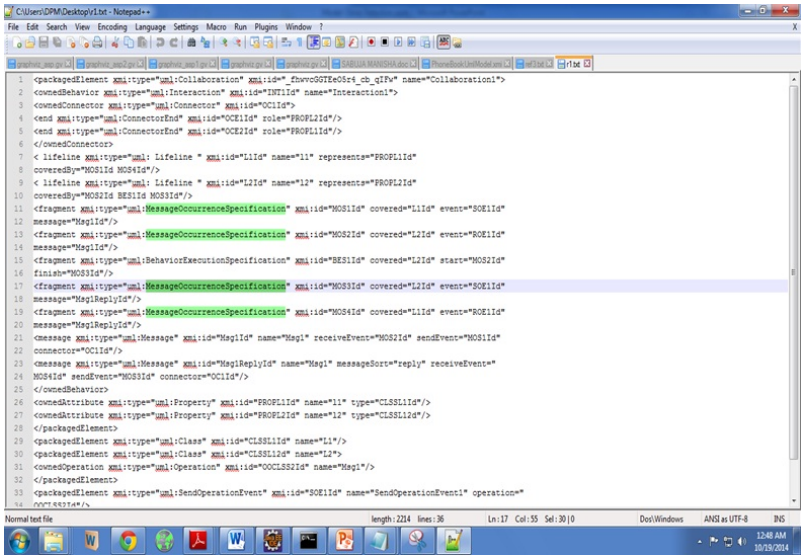

Figure 4.2: XMI representation of sequence diagrams

4.1.2 Create the Contextualized Conversations using TXL

The restructuring or the transformation process consists of the following 3 steps: Identify, Consolidate and Contextualize as shown in Figure 4.1.

Identify

This step identifies all the Behaviour Execution Specification(BES) elements in the model representation and restructures them into BES units identified by <BES>...</BES> tags. In the simple example shown in Figure 4.3, there is only one BES elements.

Consolidate

Once the tag container for the BES is created, we gather and nest all of the BES's conversation elements into the container. Each BES element has a start and finish attribute, the ids of which specify the elements of the flat representation that

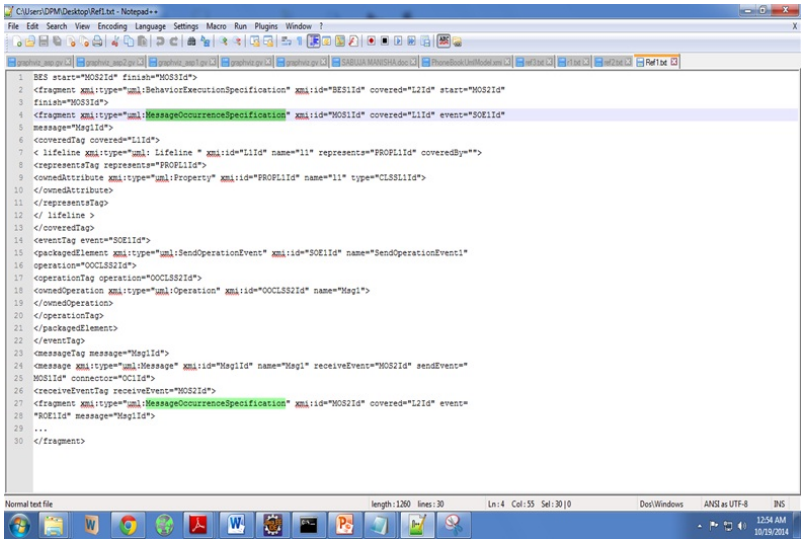

Figure 4.3: Contextualization

begin and end the BES's conversation. Due to the general ordering of message, behaviour and event occurrences in the XMI representation, this step primarily involves moving the adjacent elements of the conversation before (represents the message occurrence that start the conversation) and after the BES element until the element that represents the end of the conversation into the <BES>...</BES> unit to consolidate the whole conversation.

Contextualize

Consolidated BES conversations may consist of embedded BESs, Message Occurrence Specifications (MOSs) and Execution Occurrence Specifications (EOSs) describing the conversation?s interactions with other lifelines. Similar to BESs themselves, these use their XML attributes to link to the elements such as messages, types and other lifelines that describe their meaning. The purpose of contextualization phase is to bring all the elements referenced by the attributes into the self contained unit. Contextualization proceeds recursively for these in-lined elements. In-lining the elements of each BES in this way creates a set of self-contained interaction units for comparison during clone detection.

4.1.3 Extract Conversations using TXL

The above three steps represent the main idea of creating a granularity to compare the potential clones existing in basic SD models. An extractor module written for NiCad extracts all potential clones identified by the BES units from the contextualized input file given to NiCad. For this purpose, the general grammar for the SD representation is specialized to define the newly created BES units. Early clone detection at this stage resulted in no clones being extracted even at a 30% difference threshold. Lowering the threshold further to 35% did report clones; however, on analysing the reported clones,some were found to be undesired at the 35% threshold. They were reported as clones due to the similarity in the various elements in the contextualized unit at the bigger difference threshold. Thus it was deemed necessary to lower the threshold and perform some filtering and normalization.

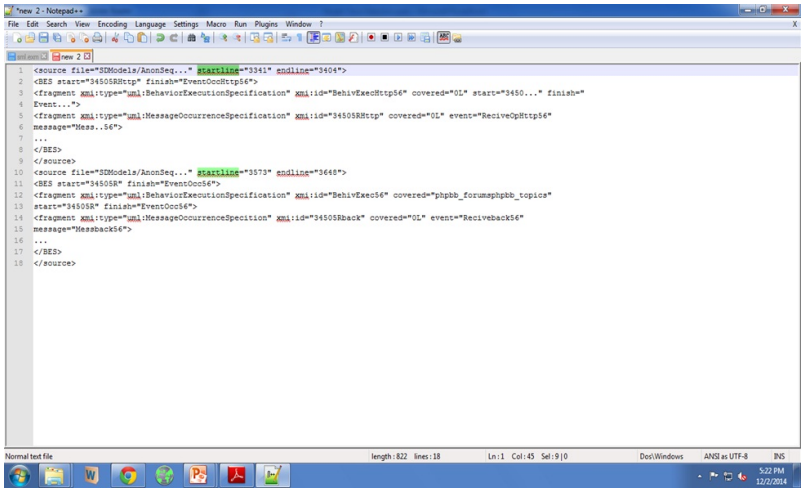

Figure 4.4: Extracted Conversations

In order to improve the precision and recall of our clone results, it was required to take a set of filtering steps to remove irrelevant elements from the comparison.

4.1.4 Detect and analyze the clones using NiCad

Here, we discuss the results obtained from clone detection on the reverse engineered SD models. We first provide results without normalization followed by a careful analysis of results after normalization. Most clone detectors require a similarity threshold to be specified during clone detection. NiCad clone detector also requires you to specify the percentage of lines that can be different in a clone pair. This threshold value is referred to as difference thresholds.

Reports generated by NiCad

The contextualized model is then input to the NiCad clone detector and the results are reported in both HTML and XML formats along with a log. By default in NiCad, these formats have reports generated displaying the various clones pairs in the model along with start and end line numbers of the clones in the contextualized representation. NiCad reports clone pairs found with and without sources. Along with these another report is generated which shows the the various clones grouped into classes based on the similarity value. The clones grouped in classes are also reported with and without source of the clones (in contextualized XMI text) along with a class identifier (<classid>) assigned to each class. For our analysis we have used the XML report displaying the clone pairs both with and without source.

4.2 Summary

In this chapter,we started with drawing a simple sequence diagram in IBM Rationale Software Architect (RSA) tool introducing two nearly same BESs. The XMI representation is obtained by exporting directly the developed sequence diagram. All BESs are identified in Identify phase, then brought together in consolidation phase and then the well structured contextualized XMI is obtained by designing corresponding Txl grammar and Txl transformation rules. The NiCad clone detector extracts the contextualized similar conversations from the XMI available.

Chapter 5

Web Services Tagging by Clone Detection

In this chapter we explain subsequent steps to find out the similarities between web services which further helps us Tagging the web services.

5.1 Proposed Methodology

We have already seen an operation definition spread on the whole WSDL document. Here, we will portray our system to combine these slices into a single limited whole that can be all the more viably associated with different operations to find service duplicity.

In forming a cluster of consolidated service operations from a WSDL description, Nicad utilizes the source transformation framework TXL. Nicad meets expectations gathering the scattered slices together to frame independent consolidated operation descriptions clustering that unmistakably and straightforwardly recognize inputs, outputs and conceivable faults. We can think about our technique as comprising of 5 dynamic stages, depicted underneath using the "SearchItem" operation in the following section.

5.1.1 Operations Extraction

The first step in the consolidation is to build up a framework for each operation. This works as a base in which to coordinate the contextual elements (such as messages and types) from their definitions in the other segments, and thus localize the semantics of the operations. Every operation from the port-Types segments, which possesses the operations inputs, outputs, and faults are extracted. Figure 5.1 shows an example of one such operation framework derived from our considered shopping mall management system example.

```
<operation name="SearchItem">
<input message="tns1:SearchItemRequest"/>
<output message="tns1:SearchItemResponse"/>
<fault message="tns1:ItemNotfoundException"/>
</operation>
```

Figure 5.1: Operations extraction

5.1.2 Message Injection

Each of the operation's inputs, outputs, and faults alludes to a named message whose definite depiction shows up somewhere else in the service description. In the following step of our combination, we resolve these message references by discovering the message portrayal for every message referred to in an input, output or fault and extending it in the relating tag. The outcome is an upgraded operation depiction in which all messages are completely depicted inside the input, output or fault that refers to them, as demonstrated in Figure 5.2.

```
<operation name="SearchItem" >
<input message="tns1:SearchItemRequest">
<message name="SearchItemRequest">
<part name="body" element="xsd1:SearchItemRequest"/>
</message>
</input>
<output message="tns1:SearchItemResponse">
<message name="SearchItemResponse">
<part name="body" element="xsd1:SearchItemResponse"/>
</message>
</output>
<fault message="tns1:ItemNotfoundException">
<message name="ItemNotfoundException">
<part name="body" element="xsd1:ItemNotfoundException"/>
</message>
</fault>
</operation>
```

Figure 5.2: Message injection

5.1.3 Element Injection

Every message consists of several parts which are also related with specific data element. WSDL Types section contains different description of those elements. In the next step of our consolidation injection of element description is carried out to their respective tags. In this way part description is the outcome with their element configuration embedded in combined message descriptions as shown in Figure 5.3. Expanding one level after another the inter-dependency of the operation definitions on its perspective can be reduced after locating the required evidences to get and evaluate it autonomously.

5.1.4 Element Consolidation

Every element is either an atomic or a complex type mentioned in a scatter manner in the service description. In consolidation phase, we discover each elements' type definition, if that is really present. For every specified type like it, we extend the definition and elements of the type and combine them into the parent element of the part. Until all elements in the part have been expanded to elements of atomic types (string, int), or there no similar type definition is present in the file, the same process is performed iteratively for all the elements of the extended type.. The outcome of

```
<operation name="SearchItem" >
<input message="tns1:SearchItemRequest">
<message name="SearchItemRequest">
<part name="body" element="xsd1:SearchItemRequest">
<element name="SearchItemRequest">
<complexType>
<sequence>
<element name="payment" type="tns1:Payment"/>
<element name="item" type="tns1:item"/>
</sequence>
</complexType>
</element>
</part>
</message>
</input>
. . .
</operation>
```

Figure 5.3: Element injection

this phase will look like the Figure 5.4.

```
<operation name="SearchItem" >
<input message="tns1:SearchItemRequest">
<message name="SearchItemRequest">
<part name="body" element="xsd1:SearchItemRequest">
<element name="SearchItemRequest">
<complexType>
<sequence>
<element name="payment" type="tns1:Payment">
<element name="ccNumber" type="xsd:int"/>
<element name="cardHolder" type="xsd:string"/>
<element name="expiryDate" type="xsd:date"/>
<element name="PIN" nillable="true" type="xsd:int"/>
</element>
<element name="Item" type="tns1:Item">
<element name="itemID" type="xsd:int"/>
<element name="itemColor" type="xsd:string"/>
</element>
</sequence>
</complexType>
</element>
</part>
</message>
</input>
. . .
</operation>
```

Figure 5.4: Element consolidation

```
<operation name="SearchItem" >
<input message="tns1:SearchItemRequest">
<message name="SearchItemRequest">
<part name="body" element="xsd1:SearchItemRequest">
<element name="SearchItemRequest">
<element name="payment" type="tns1:Payment">
<element name="ccNumber" type="xsd:int"/>
<element name="cardHolder" type="xsd:string"/>
<element name="expiryDate" type="xsd:date"/>
<element name="PIN" nillable="true" type="xsd:int"/>
</element>
<element name="Item" type="tns1:Item">
<element name="itemID" type="xsd:int"/>
<element name="itemColor" type="xsd:string"/>
</element>
</element>
</part>
</message>
</input>
<output message="tns1:SearchItemResponse">
<message name="SearchItemResponse">
<part name="body" element="xsd1:SearchItemResponse">
<element name="SearchItemResponse"/>
</part>
</message>
</output>
<fault message="tns1:ItemNotFoundException">
<message name="ItemNotFoundException">
<part name="body" element="xsd1:ItemNotFoundException">
<element name="ItemNotFoundException"/>
</part>
</message>
</fault>
</operation>
```

Figure 5.5: Cleaning Up

5.1.5 Cleaning Up

Repeated tags like <complexType>, <sequence> still exists in our expansion after all the previous 4 phases, so we delete those to bring the operation definitions simpler and easier to understand and compare. The final output for our search item example is depicted, and the overall proceedings can be pictured as stated in Figure 5.5.

Now we have clean human-readable separate operations represented with their corresponding input and deserve output. We can observe from the example the SearchItem operation includes a payment object and a Item object and gives back an acknowledgment. After composing Item and payment objects, we can easily say this service needs which information. To determine the similarity among various operations more meaningfully the above expanded form is used, which may not be possible in their source context-dependent form.

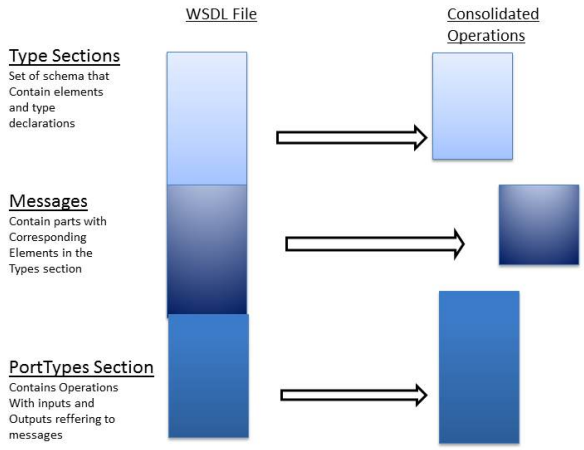

Figure 5.6: A picturization of consolidation

5.2 Results after implementation

In our work, we have examined the proposed methods with 2 sets of WSDL files collected from Seekda search engine [15, 17]. One for shopping mall management consisting of 150 web services having more than 1,000 operations, many of which are same or clones of one another. Another set for Hotel reservation systems consisting of 600 service descriptions possessing more than 7,000 operations from a large kind of domains.

In the confuguration file of NiCad Set up, the threshold is set in between 0.0 to 0.3 at different times. at the above three varying thresholds, clone detection results both on consolidated and the original non-consolidated operations are compared. The purpose behind such comparison with the help of our NiCad Tool is just to evaluate our approach whether capable of finding similarities both on consolidated and non-consolidated operations.

We have utilized the same clone detection tool NiCad on both sets of WSDL

services(Shopping Mall Management and Hotel Reservation System). NiCad has been quite successful in yielding a set of clusters comprising of similar kinds of clones at four different thresholds (0.0,0.1,0.2,0.3). The threshold indicates the piece of lines being permitted to vary between two WSDL operation descriptions assuming them to be cloned. For Example, a threshold 0.0 implies two operations are needed to be exactly same to be termed as clones, while a threshold 0.3 implies operation descriptions can have 3 lines out of 10 lines (30%) varying with 70% operation descriptions same to same.

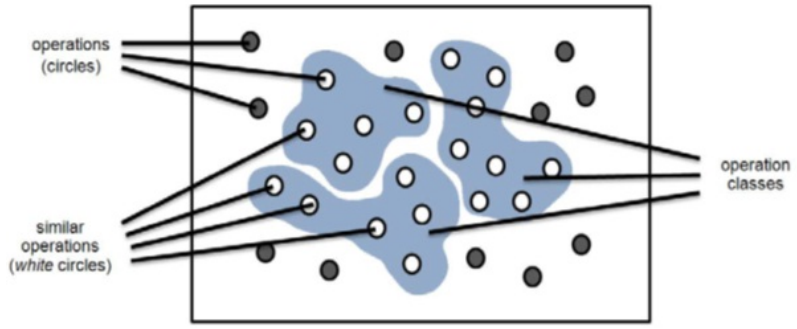

Figure 5.7: Visualization of Clone Detection

A number of matrices are considered to summarize the output, but we are going to cite here two only i.e. Number of fragments and numbers of operation classes. Number of fragments is the total number of operations having one or more existing similar operations. Operation class is the number of cluster of similar operations. Figure 5.7 shows their definitions and inter-relationship between them. A circle inside the rectangle represents an operation in the sample set. White circles inside the operation class are similar classes as shown in the figure by the shaded area surrounding them.

NiCad requires all possible classes put inside a single XML file enclosed in <source> tags, having attributes file, startline and endline indicating the original source WSDL files. NiCad also requires the starting and ending line numbers of the original operation definition inside the file.

NiCad efficiently compares every possible clones to find those clones which are differed from each other upto maximum 0.3 of threshold i.e. from exactly matched to 70% similar operation descriptions. XML files at each thresholds having a lists of <class> tags having the same operations within that particular operation class.

The operations from each of the considered 2 sets of web service descriptions are extracted and consolidated. The benefits of finding clones in consolidation approach have been analyzed comparing the results of finding clones in both consolidated and non-consolidated operations. We notice from Table 5.1 that after the consolidation, the no of clones detected reduces over the non-consolidated operations. The same phenomenon continues for all of the afore mentioned thresholds(0.0,0.1,0.2,0.3). Consolidation is considered to be desirable despite of its poor clone detection results also, as it adds comprhesion and meaning to the presentation.

Table 5.1: Number of similar (i.e. cloned) operations.

	Set1(Shpping Mall Management System)		Set2(Hotel reservation system)	
Difference Threshold	Non-Consolidated	Consolidated	Non-Consolidated	Consolidated
0.0	674	552	1332	1012
0.1	662	585	1329	1188
0.2	705	668	1331	1552
0.3	752	745	1368	1548

If two operations are the same non-consolidated, consolidating them in a same manner but separately will just include new information that is also the equal for

Table 5.2: Number of groups of similar operations

	Set1(Shpping Mall Management System)		Set2(Hotel reservation system)	
Difference Threshold	Non-Consolidated	Consolidated	Non-Consolidated	Consolidated
0.0	152	178	548	409
0.1	149	128	536	441
0.2	180	148	566	635
0.3	178	145	558	628

Figure 5.8: Input WSDL to the parser

both. But the scenario is not like it . Let's take two non-consolidated operations having same input and output tags. We observe that they use unmatched types which may even have totally separate name by investing the type definitions of the parameters,though They may seem to be the same. In Figure 5.1, we see the description of two non- consolidated operations said to be exactly the duplicate. Anyway, We will find that they contextualize to take much different meanings,if we consolidate them based on their context . Mainly, we see that the term Stock is used to indicate inventory in one perspective and a financial stock quote in the other.

The second observation of ours is that operation classes can be fragmented with the help of consolidating the operation. For Example, a class containing 'n' consolidated operations may be divided in two classes of 'n/2' consolidated operations in each.

Finally, and most significantly, we observe that without consolidating it would not have been possible on part of us to discover duplicate operations with similarity detection. For example, let's take the operation class shown in Figure 5.7. It

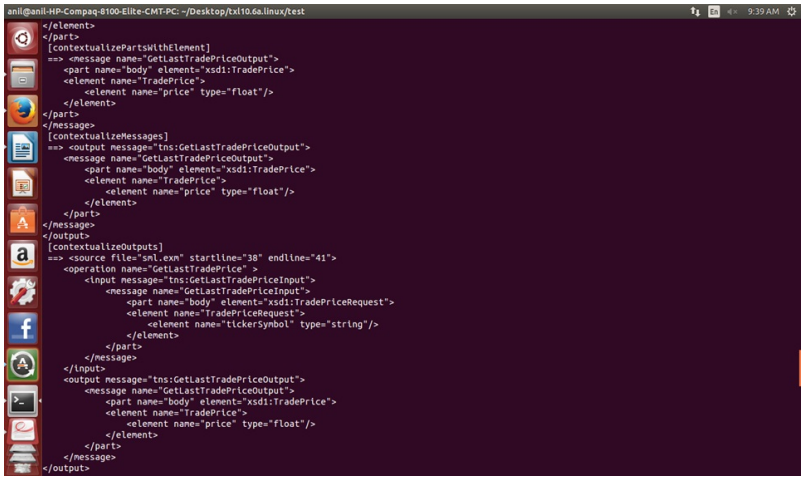

Figure 5.9: Contextualizing the WSDL using Txl program

has operations, all linked to charting, with absolutely unlike names that could not be identified as linked by clone detection without consolidation. Some, such as DrawYieldCurveCustom, don't even contain chart in their name. The motive these are identified as being same by NiCad is that the consolidation extends all type definitions into the operation description, and all of these operations really possess very similar elements.

We hope that examples like this authenticate our primary faith that consolidating operation descriptions can permit us to more efficiently find duplicate operations. It contextualizes another bare operation description, which permits fundamental clone detection tools like NiCad to find exact basing on the context as shown in Figure 5.9.The output after parsing lools like Figure 5.10.

5.3 Summary

WSDL descriptions of web services impose difficulty for discovering clones. In this work we have explained how we can influence code clone detection methodologies to spot and gather same kind of services by reconfiguring service descriptions to consolidate inaccessible information. Our consolidation converts WSDL descriptions

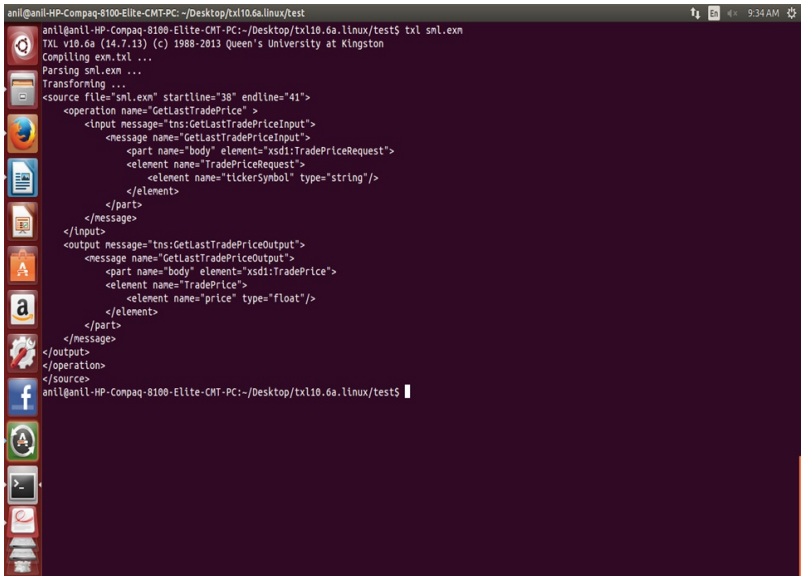

Figure 5.10: Output Generated after parsing

human accessible and responsive to analysis using code duplication methodologies bu adding simplicity. The consolidation not only improves the output yield by clone detection to escape recognizing services that while same on the surface really have very unlike semantics, but also generates outputs recognizing duplicate services that could not be detected without consolidation.

Chapter 6

Conclusion and Scope for Future Work

6.1 Conclusion

The work in this thesis, comes out of our interest in finding how NiCad can be applied to find clones in UML behavioral models, called Sequence Diagrams (SD). We work with reverse engineered SD models in their XMI representation. The models represent the execution traces of different users in various access control roles, navigating the popular open source phpBB forum. The XMI representation of the SD models has a flat structure with no immediate granularity visible to be used as the unit of comparison for clone detection. Thus it became necessary to identify a level of granularity for the SDs. Upon analyzing this representation, we found that all the Occurrence Specifications for the messages and the Behavioral Execution Specification are ordered in the sequence diagram model. Since the duration for the execution of a behavior is represented by the Behavioral Execution Specification element, it led us to propose and automate the creation of a Behavioral Execution Specification unit (BES unit) to be the granularity that is required for clone comparison. We interchangeably refer to this unit as a conversation .

WSDL descriptions of web services pose problems for finding similarities using clone detectors like NICAD. Operation descriptions are broken up into pieces and delocalized making it difficult to extract appropriate units to compare. If just one type of piece is extracted, then the others are unavailable later in the clone analysis. This led us to propose a NICAD Pre-Transformer to make transformations of the original code base possible before the extraction of potential clones. This allows operation descriptions to be reorganized into a set of consolidated units that can then be extracted and compared. In the absence of such a phase, we developed a special WSDL extractor for NICAD that performs this consolidation while extracting operations.

6.2 Scope for Further Research

We can take our thesis in following several ways-

First, as Sequence Diagram is not the only UML diagram to be considered for finding duplicate fragments in them. Class diagram considering Object constraint languages (OCL) ans CRC card is considered to be the most accurate one in presenting the system functionality. So this thesis can be improved to find duplicates in class diagram also.

Second, there are numerous normalizations with which we can experiment in NICAD to possibly achieve better results. For example, we could remove closing tags since they provide no meaningful information for clone detection and they can even be harmful to the results. Since most WSCells share at least 5 closing tags at the end due to nesting, this can lead to false positives, particularly for smaller WSCells. Other possible normalizations include, putting attributes like name and type on their own line, removing repetitive tags (e.g. <message> and <part>), and even going as far as to remove all tags (i.e. using only attributes) while maintaining order.

Bibliography

[1] C. K. Roy and J. R Cordy, *Nicad: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization*, In ICPC 2008, The 16th IEEE International Conference on, pp. 172181., IEEE, 2008.

[2] H. Liu, Z. Ma, Lu Zhang, and W. Shao, *Detecting duplications in sequence diagrams based on suffix trees*, In APSEC, pp. 269276, 2006.

[3] D. Martin and J.R. Cordy, *Analyzing Web Service Similarity Using Contextual Clones*, In Proceedings of the 5th International Workshop on Software Clones, IWSC 2011, Waikiki, Hawaii, pp. 41-46, May 2011.

[4] X. Dong, A. Halevy, J. Madhaven, E. Nemes and J. Zhang, *Similarity Search for Web Services*, In Proceedings of the 30th VLDB Conference, Toronto, Canada, pp. 372-383,August 2004,

[5] D. Rattan, R. Bhatia, and M. Singh, *Model clone detection based on tree comparison,* in INDICON, 2012.

[6] M. H. Alalfi, J. R. Cordy, and T. R. Dean. *Automated verification of role-based access control security models recovered from dynamic web applications.* In WSE, pages 110, 2012.

[7] H. Strrle. *Towards clone detection in uml domain models.* Software and System Modeling, 12(2):307329, 2013.

[8] R. Komondoor and S. Horwitz, *Using Slicing to Identify Duplication in Source Code,* Proc. Intl Symp. Static Analysis, pp. 40-56, July 2001.

[9] F. Gauthier, T. Lavoie, and E. Merlo. *Uncovering access control weaknesses and flaws with security-discordant software clones.* In ACSAC, pages 209218, 2013.

[10] E. Stroulia and Y. Wang, *Structural and Semantic Matching for Assessing Web Service Similarity*, International Journal of Cooperative Information Systems 14,4, pp. 407-437, 2005.

[11] J.R. Cordy, *The TXL Source Transformation Language*, Science of Computer Programming. 61,3, pp. 190-210, 2006.

[12] M. P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann: *Service-Oriented Computing: State of the Art and Research Challenges, Computer*, VOL. 40, NO. 11, pp. 38-45 2007.

[13] N. Milanovic, M. Malek: *Current solutions for web service composition.* IEEE Internet Computing 8 (2004) 5159 ,2004

[14] F. Curbera, R. Khalaf, N. Mukhi, S. Tai, S. Weerawarana: *The next step in web services.* Communications of the ACM 46 ,2003, 2934 ,2003 bibitemloiH. Meyer, M. Weske: *L*ight-Weight Semantic Service Annotations through Tagging, ICSOC 2006

[15] M. Li, J. Zhao, L. Wang, Sibo Cai, B. Xie: CoWS: *An Internet-Enriched and Quality-Aware Web Services Search Engine,* ICWS 2011

[16] WSDL, http://www.w3.org/TR/wsdl

[17] UDDI, http://uddi.xml.org/

[18] Seekda, http://webservices.seekda.com/

[19] Service-Finder, http://demo.service-finder.eu/

[20] Wikipedia, http://en.wikipedia.org/wiki/$Tag_(metadata)$